Model Checking as Control: Feedback Control for Statistical Model Checking of Cyber-Physical Systems

K Kalajdzic, Cyrille Jegourel, E Bartocci, Axel Legay, Scott Smolka, R Grosu

Model Checking as Control: Feedback Control for Statistical Model Checking of Cyber-Physical Systems

K. Kalajdzic[1], C. Jegourel[2], E. Bartocci[1],
A. Legay[2], S.A. Smolka[3], and R. Grosu[1]

[1] Vienna University of Technology, Austria
[2] INRIA Rennes, Bretagne Atlantique, France
[3] Stony Brook University, NY, USA

Abstract We introduce *feedback-control statistical system checking (FC-SSC)*, a new approach to statistical model checking that exploits principles of feedback-control for the analysis of cyber-physical systems (CPS). FC-SSC uses stochastic system identification to learn a CPS model, importance sampling to estimate the CPS state, and importance splitting to control the CPS so that the probability that the CPS satisfies a given property can be efficiently inferred. We illustrate the utility of FC-SSC on two example applications, each of which is simple enough to be easily understood, yet complex enough to exhibit all of FC-SCC's features. To the best of our knowledge, FC-SSC is the first statistical system checker to efficiently estimate the probability of rare events in realistic CPS applications or in any complex probabilistic program whose model is either not available, or is infeasible to derive through static-analysis techniques.

1 Introduction

Modern distributed systems, and *cyber-physical systems* (CPSs) in particular, embed sensing, computation, actuation, and communication within the physical substratum, resulting in open, probabilistic, systems of systems. CPS examples include smart factories, transportation systems, and health-care systems [4].

Openness, uncertainty, and distribution, however, render the problem of *accurate prediction* of the (emergent) behavior of CPSs extremely challenging. Because of (exponential) state explosion, model-based approaches to this problem that rely on *exhaustive state-space exploration* such as classical model checking (MC) [5], are ineffective. *Approximate prediction* techniques, such as *statistical model checking* (SMC), have therefore recently become increasingly popular [10,22,6]. The key idea behind SMC is to sample the model's execution behavior through simulation, and to use statistical measures to predict, with a desired confidence and error margin, whether the system satisfies a given property. An important advantage of SMC is that the sampling can be parallelized, thus benefiting from recent advances in *multi-core* and *GPU* technologies [2].

A serious *obstacle* in the application of SMC techniques is their poor performance in predicting the satisfaction of properties holding with very low probability, so-called *rare events* (REs). In such cases, the number of samples required

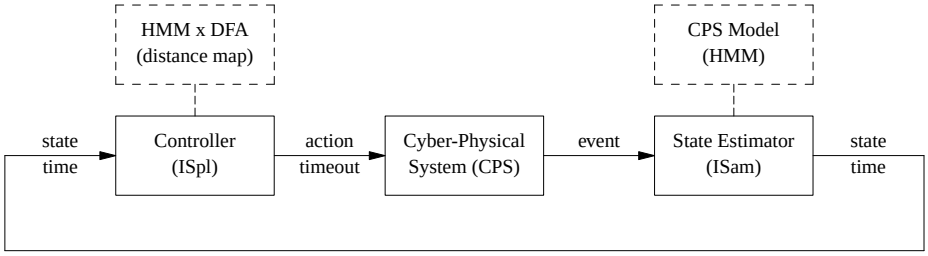

Figure 1: FC-SSC as a feedback controller exploiting ISam and ISpl.

to attain a high confidence ratio and a low error margin explodes [23,10]. Two sequential Monte-Carlo techniques, *importance sampling (ISam)* [7] and *importance splitting (ISpl)* [9], originally developed for statistical physics, promise to overcome this obstacle. These techniques have recently been adopted by the robotics [21,19] and SMC communities [23,20,15,11,12].

ISpl and ISam have individually demonstrated their *utility* on a number of models. We are still, however, *a long way from the statistical checking (SC) of CPSs*. In particular, the following three challenges have not yet been addressed:

1. *The CPS model is generally not known*, as either the basic laws of the substratum, or the control program, are only partially available. Consequently, a finite-model abstraction through static analysis is infeasible.
2. *The CPS state is generally not known*, as either the output represents only a small fraction of the set of state variables, or the output represents an arbitrary function defined on a subset of the state variables.
3. *The CPS steering policy towards REs is generally not known*, as the system model is not available in advance, and consequently, the relationship between the RE property and the CPS behavior is not known as well.

In this paper, we attack these three challenges by proposing a novel *feedback-control framework for the SC of CPSs (FC-SSC)*; see Figure 1. To the best of our knowledge, this is the first attempt to define SC as control and to completely automate RE estimation in CPSs. In FC-SSC, we automatically:

1. *Learn the CPS model*. We assume that we *can observe the CPS outputs*, which are either measurements of the physical part or values output by the cyber part. Using a (learning) set of observation sequences and statistical system-identification (machine-learning) techniques [18], we automatically *learn a hidden Markov Model* (HMM) of the CPS under investigation.
2. *Infer the CPS state*. Having access to the current observation sequence and the learned HMM, we employ statistical inference techniques to determine the hidden state [18]. To scale up the inference, we use ISam as an approximation algorithm. Although ISam was originally introduced for rare-event estimation, its practical success is in state estimation.
3. *Infer the CPS control policy*. We assume that we *can start the CPS from a given state, and run it for a given amount of time*. In order to steer the

system towards an RE, we use ISpl. This requires, however, an RE decomposition into a set of *levels*, s.t. the probabilities of going from one level to the next are essentially equal, and the product of these inter-level probabilities equals the RE probability. By using the learned HMM and the RE property, we automatically derive an optimal RE decomposition into levels.

In FC-SSC, ISam estimates the current CPS state and the current level, and ISpl controls the execution of the CPS based on this information. Both techniques depend on the HMM identified during a preliminary, learning stage. FC-SSC may be applied to the approximate analysis of any complex probabilistic program whose: 1) Monitoring is feasible through appropriate instrumentation, but whose 2) Model derivation is infeasible through static analysis techniques (due to e.g. sheer size, complicated pointer manipulation).

The rest of the paper is organized as follows. In Section 2, we introduce two running examples, simple enough to illustrate the main concepts, while still capturing the essential features of complex CPSs. In Section 3, we introduce our learning algorithm, based on expectation maximization [18]. In Section 4, we present our ISam-based state-estimation algorithm, while in Section 5, we present our ISpl-based control algorithm. In Section 7, we discuss the results we obtained for the two example systems. Finally, in Section 8, we offer our concluding remarks and discuss future work.

2 Running Examples

In order to illustrate the techniques employed in FC-SSC, we use as running examples two simple (but not too simple) probabilistic programs: Dining Philosophers and the Incrementer-or-Resetter program.

Dining Philosophers. This example was chosen because its model is very well known, its complexity nicely scales up, and its rare events are very intuitive. Moreover, the multi-threaded program we use to implement Dining Philosophers illustrates the difficulties encountered when trying to model check real programs, such as their interaction with the operating system and their large state vector. In classic model checking, the former would require checking the associated operating-system functions, and the latter would require some cone-of-influence program slicing. Both are hard to achieve in practice.

For monitoring purposes however, all that one needs to do is to instrument the entities of interest (variables, assignments, procedure calls, etc.) and to run the program. Extending monitoring to SSC requires however an HMM, a way of estimating the hidden states, and a way to control the program. Our code is based on the variant of randomized Dining Philosophers problem without fairness assumption, introduced in [8]. To minimize the interference of instrumentation with the program execution, we instrument only one thread. To account for the unknown and possibly distinct executions of the uninstrumented part of the program, we add loops (do_some_work) whose execution time is distributed, for simplicity, according to a uniform probability distribution.

```
void *philosopher(int this) { while (true) {
    do_some_work();
    switch (phil_state[this]) {
    case 0: /* cannot stay thinking so move to trying */
        phil_state[this] = 1; break;
    case 1: /* draw randomly */
        if (flip_coin() == COIN_HEADS) phil_state[this] = 2;
        else phil_state[this] = 3; break;
    case 2: /* try to pick up left fork */
        emit_symbol(SYM_TRY); pthread_mutex_lock(&fork[this]);
        if (fork_state[this] == FORK_FREE)
          phil_state[this] = 4; fork_state[this] = FORK_TAKEN;
        pthread_mutex_unlock(&fork[this]); break;
    case 3: /* try to pick up right fork */
        emit_symbol(SYM_TRY); pthread_mutex_lock(&fork[(this + 1) % n_phil]);
        if (fork_state[(this + 1) % n_phil] == FORK_FREE)
          phil_state[this] = 5; fork_state[(this + 1) % n_phil] = FORK_TAKEN;
        pthread_mutex_unlock(&fork[(this + 1) % n_phil]); break;
    ...
    case 9: /* eat */
        emit_symbol(SYM_EAT);
        if (flip_coin() == COIN_HEADS) {
          pthread_mutex_lock(&fork[this]); phil_state[this] = 10;
          fork_state[this] = FORK_FREE; pthread_mutex_unlock(&fork[this]);}
        else {
          pthread_mutex_lock(&fork[(this + 1) % n_phil]); phil_state[this] = 11;
          fork_state[(this + 1) % n_phil] = FORK_FREE;
          pthread_mutex_unlock(&fork[(this + 1) % n_phil]);} break;
    case 11: /* drop left fork */
        emit_symbol(SYM_DROP_FORKS); pthread_mutex_lock(&fork[this]);
        phil_state[this] = 0; fork_state[this] = FORK_FREE;
        pthread_mutex_unlock(&fork[this]); break;
    default: fatal_error("incorrect philosopher state"); }}}
```

Figure 2: C code snippet of the main loop in the Dining Philosophers

For space reasons, we show in Figure 2 only a snippet of the C-code of the main loop of a philosopher. The full code is available from [1]. As it is well known, each philosopher undergoes a sequence of modes, from thinking, to picking one fork, then the other, eating and then dropping the forks. It may drop the single fork it holds also when it cannot pick up the other fork. Given, say, 100 philosophers, the RE in this case is the property that a particular philosopher k succeeds to eat within a given interval of time.

Incrementer-or-Resetter. In contrast to Dining Philosophers, this is a nested, sequential program, notoriously difficult to analyze with simplistic SMC. The program aims to increment a counter variable up to a value N, within two nested loops. At each step however, the program uniformly chooses to either increment the counter or reset it to zero. The RE in this case is achieving the final value N for the counter. The C code of a particular instance of the program is shown in

```
int main(void) {
    int i, j, counter = 0;
    for (i = 0; i < 7; i++) {
        for (j = 0; j < 4; j++) {
            counter = random() % 2 ? counter + 1 : 0; }
        /* instrumented to emit counter value */
        printf("%d\n", counter / 4); }
    return 0; }
```

Figure 3: C program implementing the Incrementer-or-Resetter

Figure 3. In this proof-of-concept example, there are 28 steps, in each of which the counter is either incremented or reset to zero.

As is common programming practice, the monitoring overhead incurred by instrumentation is reduced by instrumenting the outer loop only. Moreover, by dividing the counter value by 4, we also reduce the number of observable events. This further simplifies the analysis and the learning process.

3 System Identification

We assume that the CPS models are finite state. This captures the influence of the cyber part on the CPS. For simplicity, we also assume that the models have only one state variable, that is, they are *Hidden Markov Models* (HMMs) [19]. Note, however, that all the techniques introduced in this and the following sections work as well for continuous-state linear Gaussian models [18].

An HMM defines two sequences, $X_1X_2\ldots X_t$ and $Y_1Y_2\ldots Y_t$, over time, where X_t and Y_t are the random state and output variable at time t, respectively. The values x_t of X_t and y_t of Y_t range over the finite sets Σ and Υ, respectively. Since in an HMM X_{t+1} only depends on X_t, and Y_t only depends on X_t, the HMM can be concisely represented by three probability distributions (PDs):

- $\pi = P(X_1)$, the prior *initial state* PD,
- $A = P(X_{t+1}|X_t)$, the conditional *next state* PD,
- $C = P(Y_t|X_t)$, the conditional *output* PD.

Equivalently, an HMM consists of a triple $H = (\pi, A, C)$, where π is a probability vector of dimension $N = |\Sigma|$ having an entry for each $P(X_1 = x_1)$, and A and C are probability matrices of dimensions $N \times N$ and $N \times |\Upsilon|$, respectively. Given N, Υ, and observation sequence $\overline{y} = y_1y_2\ldots y_T$, the goal of *system identification* is to learn the HMM $H = (\pi, A, C)$, maximizing the expectation that an execution sequence $\overline{x} = x_1 \ldots x_T$ of H produces output $\overline{y}$.

The algorithm is therefore known as the *expectation-maximization (EM)*, or *Baum-Welch (BW)* (after its authors) algorithm [17,18]. Maximizing the expectation as a function of H is equivalent to maximizing:

$$\mathcal{L}(H) = \log P(\overline{y}|H) = \log \textstyle\sum_x P(x, \overline{y}|H) = \log \sum_x Q(x)(P(x, \overline{y}|H)/Q(x))$$

where $Q(x)$ is an arbitrary distribution over the state variable. Using Jensen's inequality and expanding the division within the logarithm one obtains:

$$\mathcal{L}(H) \geq \textstyle\sum_x Q(x) \log P(x, \overline{y}|H) - \sum_x Q(x) \log Q(x) = \mathcal{F}(H, Q)$$

The EM algorithm now alternates between two maximization steps:

$$\text{E}-\text{step}: Q_{k+1} = \text{argmax}_Q\ \mathcal{F}(H_k, Q) \quad \text{M}-\text{step}: H_{k+1} = \text{argmax}_H\ \mathcal{F}(H, Q_k)$$

The E-step is maximized when $Q_{k+1}(x) = P(X = x \,|\, \overline{y}, H_k)$, in which case likelihood $\mathcal{L}(H_k) = \mathcal{F}(H_k, Q_{k+1})$. The M-step is maximized by maximizing the first term in $\mathcal{F}(H, Q)$, as the second (the entropy of Q) is independent of H [18]. Computing $P(X = x \,|\, \overline{y}, H)$ is called *filtering*, which for HMMs, takes the form of the *forward-backward* algorithm. Maximizing the M-step also takes advantage of filtering, as shown in algorithm Learn below. Let:

$$\alpha_i(t) = P(\overline{y}_{1:t}, X_t = i \mid H) \qquad \beta_i(t) = P(\overline{y}_{t+1:T} \mid X_t = i, H)$$
$$\gamma_i(t) = P(X_t = i \mid \overline{y}, H) \qquad \xi_{ij}(t) = P(X_t = i, X_{t+1} = j \mid \overline{y}, H)$$

Then the system-identification algorithm Learn is defined as in Algorithm 1.

Algorithm 1: HMM Learn ($\overline{y}$, N, Υ, ϵ)

initialize $H^* = (A, C, \pi)$ randomly
repeat
  $H = H^*$;
  (* E-Step *)
  $\alpha_i(1) = \pi_i c_i(y_1)$; $\alpha_i(t) = c_i(y_t) \sum_{j=1}^{N} \alpha_j(t-1) a_{ji}$; $\forall i{=}1{:}N, t{=}2{:}T$ //Fwd
  $\beta_i(T) = 1$; $\beta_i(t) = \sum_{j=1}^{N} \beta_i(t+1) a_{ij} c_j(y_{t+1})$; $\forall i{=}1{:}N, t{=}1{:}T{-}1$ //Bwd
  $\gamma_i(t) = \alpha_i(t)\beta_i(t) / \sum_{j=1}^{N} \alpha_j(t)\beta_j(t)$; $\forall i{=}1{:}N, t{=}1{:}T$ //Fwd-Bwd
  $\xi_{ij}(t) = \alpha_i(t) a_{ij} \beta_i(t+1) c_j(y_{t+1}) / \sum_{k=1}^{N} \alpha_k(t)\beta_k(t)$; $\forall i, j{=}1{:}N, t{=}1{:}T$
  (* M-Step *)
  $\pi_i^* = \gamma_i(1)$; $\forall i{=}1{:}N$
  $a_{ij}^* = \sum_{t=1}^{T-1} \xi_{ij}(t) / \sum_{t=1}^{T-1} \gamma_i(t)$; $\forall i, j{=}1{:}N$
  $c_{iy}^* = \sum_{t=1}^{T} 1_{y_t = y} \gamma_i(t) / \sum_{t=1}^{T} \gamma_i(t)$; $\forall i{=}1{:}N, y \in \Upsilon$
until ($\mathcal{L}(H^*) - \mathcal{L}(H) \leq \epsilon$);
return (H^*)

For the Dining Philosophers, we have collected a number of long traces emitted by a single thread and used them to learn a 6-state HMM shown in Figure 4. For the Incrementer-or-Resetter example, the traces were produced by emitting the value of the counter divided by 4 (see Figure 3). We thus have a single symbol c representing the counter values in the interval $[4c, 4c+3]$, with $c \in \{0, 1, 2, 3, 4, 5, 6\}$. The HMM we learned for this example contains 4 states and has 7 symbols for all the possible values of c. It is shown in Figure 5.

4 State Estimation

Algorithm 1 uses the entire observation sequence $\overline{y}$ to *a posteriori* compute the probability $P(X_t = i \,|\, \overline{y}, H)$. If, however, one has the observation $\overline{y}$ only up to time $T = t$, this becomes a *forward state-estimation* algorithm:

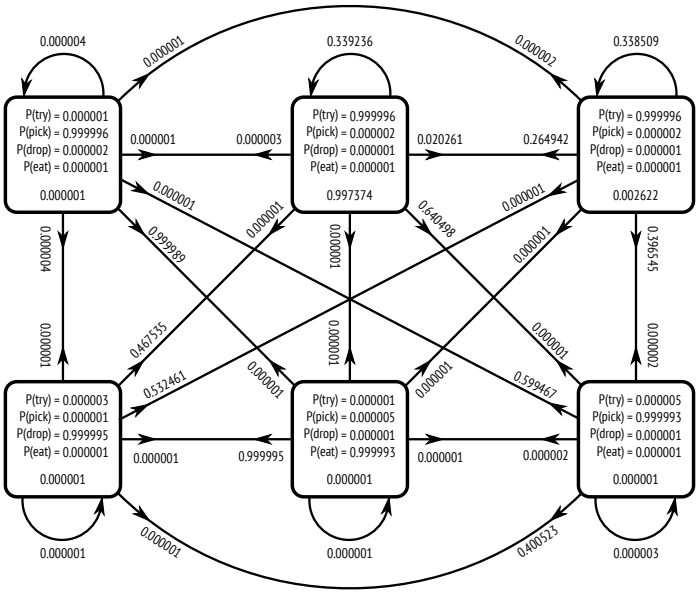

Figure 4: HMM modeling a single thread of the Dining Philosophers program.

$$P(X_t = i \,|\, \overline{y}, H) = \alpha_i(t) / \textstyle\sum_{j=1}^{N} \alpha_j(t) \quad \forall i{=}1{:}N$$

In practice, this algorithm may be inefficient, and an approximate version of it based on *importance sampling* (ISam) is preferred. The key idea is as follows. Each sample, also called a *particle*, takes a random transition from its current state $X_t = i$ to a next state $X_{t+1} = j$ according to a_{ij}. Its importance (weight) $c_j(y_{t+1})$ is thereafter used in a resampling phase which discards particles that poorly predicted y_{t+1}. ISam is therefore a *particle filtering* algorithm.

Initially distributing the K particles according to π confers on ISam two salient properties: 1) The K particles are always distributed among the most promising states; and 2) When K approaches infinity, the probability $P(X_t = i \,|\, \overline{y}, H)$ is accurately estimated by the average number of particles in state i.

In addition to the HMM H identified as discussed in Section 3, we also assume that the RE property of interest is given as a *deterministic finite automaton (DFA)* $D = (s_o, B, F)$ where $s_0 \in S$ is the *initial state*, B is the *transition function* from $S \times \Upsilon \to S$, and $F \subseteq S$ is the set of *accepting states*.

The DFA D accepts the output of the HMM H as its input, and it is run as a consequence in conjunction with H. Formally, this corresponds to the parallel composition of H and D as shown in Algorithm 2. This composition is used by ISpl to determine the levels used by the control algorithm.

Algorithm 2: Estimate (K, H, D)

$x_i = \text{sample}(\pi)$; $\quad s_i = s_0$; $\quad w_i = 1$; $\quad \forall i = 1{:}K$

while (true) **do**
  on y **do** $(\overline{x}, \overline{s}, \overline{w}) = \text{nextEstimate}(K, y, \overline{x}, \overline{s}, \overline{w}, A, B, C)$;

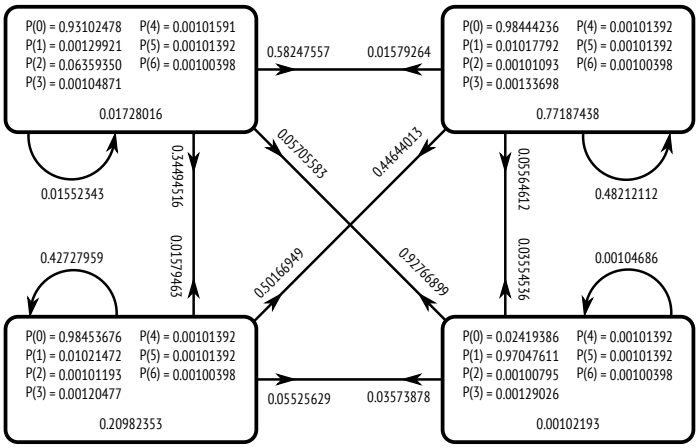

Figure 5: HMM modeling the Incrementer-or-Resetter program.

The input to Estimate is the number of particles K, the HMM H, and the DFA D. Its local state is a configuration of particles $(\overline{x}, \overline{s}, \overline{w})$, containing for each particle i, the state x_i in the HMM, the state s_i in the DFA, and a weight w_i. The initial state $\overline{x}$ is distributed according to π, the initial state $\overline{s}$ is equal to s_0, and the initial weight $\overline{w}$ is equal to 1. On every output y thrown by the CPS, Estimate calls nextEstimate to get the next particle configuration.

Algorithm 3: PC nextEstimate $(K, y, \overline{x}, \overline{s}, \overline{w}, A, B, C)$

$x_i = \text{sample}(A(x_i)); \quad s_i = B(s_i, y); \quad w_i = w_i\, C(x_i, y); \quad \forall i = 1{:}K$

normalize$(\overline{w})$;

if $(1/\sum_{i=1}^{K} w_i^2 \ll K)$ **then** $(\overline{x}, \overline{s}, \overline{w}) = \text{resample}(\overline{x}, \overline{s}, \overline{w})$

return $(\overline{x}, \overline{s}, \overline{w})$

NextEstimate works as described at the beginning of this section. For each particle i, it samples the next state x_i from $A(x_i)$, computes the next state s_i as $B(s_i, y)$, and computes the next weight w_i. To improve accuracy, this weight is multiplied with its previous value. NextEstimate then normalizes $\overline{w}$ and resamples the particles if necessary. It returns the new particle configuration PC.

5 Feedback Control

Given a system model H and a safety property φ, a statistical model checker aims to estimate the probability $P(\varphi \,|\, H)$ of H satisfying φ. Moreover, it tries to do this within time T.

If φ is a *rare event* (RE), i.e., its satisfaction probability in H is very low, the *importance splitting algorithm* (ISpl) [14,9,12] seeks to decompose φ into a set of M formulas $\varphi_1 \ldots \varphi_M$, also called *levels*, such that:

$$P(\varphi \,|\, H) = P(\varphi_M \,|\, \varphi_{M-1}, H) \times \ldots \times P(\varphi_2 \,|\, \varphi_1, H)$$

Algorithm 4: Optimized adaptive levels

Let $\tau_\varphi = \min\{\Phi(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy φ
$k = 1$ $\forall j \in \{1, \ldots, N\}$, generate path ω_j^k **repeat**
  Let $Q = \{\Phi(\omega_j^k), \forall j \in \{1, \ldots, N\}\}$ Find minimum $\tau_k \in Q$ $\tau_k = \min(\tau_k, \tau_\varphi)$
  $I_k = \{j \in \{1, \ldots, N\} : \Phi(\omega_j^k) > \tau_k\}$ $\tilde{\gamma}_k = \frac{|I_k|}{N}$ $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$ **for** $j \notin I_k$
  do
    choose uniformly randomly $l \in I_k$
    $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \{\omega \in pref(\omega_l^k) : \Phi(\omega) < \tau_k\}$ generate path ω_j^{k+1} with prefix $\tilde{\omega}_j^{k+1}$
  $M = k$; $k = k + 1$
until $\tau_k > \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

and, for $k = 2{:}M$, the considerably larger probabilities $P(\varphi_k \mid \varphi_{k-1}, H)$ are essentially equal. This minimizes the variance of the product estimation.[4]

The intractable problem of model checking $P(\varphi \mid H)$ within time T is thus reduced to a set of more tractable estimation problems $P(\varphi_k \mid \varphi_{k-1}, H)$ within time T_k, such that $\sum_k T_k \leq T$.

The exact computation of $P(\varphi_k \mid \varphi_{k-1}, H)$ may still be hard. ISpl, like ISam, is therefore using an approximate particle-filtering technique. Like ISam, it starts N particles of H from level φ_{k-1}, runs them for at most $T - (\sum_{k=1}^{k-1} T_k)$ time, where the time it takes to reach the initial-state level $T_1 = 0$, and computes their scores $\Phi(\omega_j^k)$, $j = 1{:}N$, according to how close their traces ω_j^k are satisfying φ_k.

The number of particles satisfying φ_k divided by N approximates the probability $P(\varphi_k \mid \varphi_{k-1}, H)$. Moreover, through rebranching and resampling, the promising particles get multiplied, and those with the lowest scores discarded. The estimation of $P(\varphi_{k+1} \mid \varphi_k, H)$ is then initiated, starting from the current state of the surviving particles. The process continues up to φ_M.

Like ISam, ISpl always directs the particles towards the most promising parts in H, and when N tends to infinity, the estimate it computes becomes exact. ISpl thus closely resembles ISam, except for the way it computes the particle weights (which have a different meaning) and for the idea of decomposing φ.

While various decomposition ideas were presented, for example in [9,12], the *automatic derivation of* $\varphi_1 \ldots \varphi_M$, however, has so far proved elusive. Moreover, this becomes a *grand challenge if one is given a CPS* R instead of a model H. The only thing one can typically do with R is to start it from a (most often opaque) state, run it for some time T, observe during this time its output $\overline{y}$, and possibly store its last (again opaque) state for later reuse.

[4] Importance splitting has been first used in [14] to estimate the probability that neutrons would pass through certain shielding materials. The distance traveled in the shield can then be used to define a set of increasing levels $0 = l_1 < l_2 < \cdots < l_n = \tau$ that may be reached by the paths of neutrons, with the property that reaching a given level implies having reached all the lower levels.

Fortunately, as we have seen in Section 3, this is enough for identifying an HMM H of CPS R, whose dimension N is chosen such that: (1) It best reproduces $\overline{y}$; and (2) A dimension of N+1, does not significantly improve its predictions.

As seen in Section 4, the product of the HMM H with the DFA D encoding the safety property φ is a Markov chain M, whose states are marked as accepting according to D. The use of D instead of φ is with no loss of generality, as φ is a safety property, and its satisfying traces are the accepting words of D.

The states of M are computed by ISam and they can be used to compute the levels φ_k. For this purpose, we apply offline the statistical model checker of the PRISM model-checking suite (prismmodelchecker.org) to H. This is feasible since the size of H is small. In a simple and intuitive way, the level of a state s is computed as the minimum distance to an accepting state. In a more refined version, the level of s is computed as the probability of reaching an accepting state from s. Section 6 describe our scoring (leveling) algorithm in more detail.

6 Scoring

The process of computing levels for ISpl begins by an offline reachability analysis first proposed in [3]. With this approach, we first compose the system HMM H with the property DFA D to obtain a Discrete-Time Markov Chain (DTMC) M. We then formulate the problem of reaching an accepting state of the DFA as a reward-based reachability query, and finally execute the PRISM model checker to compute the expected number of steps (distance) required to reach an accepting state from any compound state (i, j) of M.

Through this reward-based bounded-reachability analysis, for each DTMC state $(i, j), i \in \{1, 2, \ldots, N_h\}, j \in \{1, 2, \ldots, N_d\}$, we calculate the distance $\delta_{i,j}$ from an accepting state. We subsequently normalize all the distances by dividing them with $max(\delta_{i,j})$ and subtract the normalized distances from 1. The result is a numerical measure of the "closeness" of every state (i, j) to the satisfaction of the property. We will call this measure a *level* and denote it as $L_{i,j}$ such that:

$$L_{i,j} = 1 - \frac{\delta_{i,j}}{max(\delta_{i,j})}$$

In a state farthest from the satisfaction of the property, $L = 0$, whereas in an accepting state, $L = 1$. Having defined the level of all the states, we can order them numerically. In the specific case of our Dining Philosophers example, after performing the PRISM reachability analysis, we obtain the ordering of states shown in Figure 6.

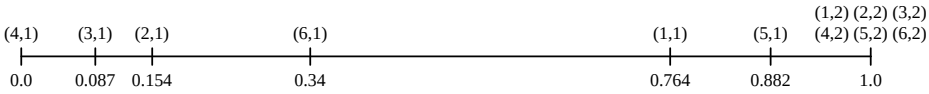

Figure 6: Compound states (i, j) of the parallel composition $H \times D$, ordered on a scale from 0 to 1 based on their potential for satisfying the property.

The levels $L_{i,j}$ are computed in advance of executing FC-SSC, and, in some cases, they might be too coarse for a good estimation of the RE probability by

ISpl. To help refine the estimation process, we use Algorithm 4, proposed by [13], which adaptively derives the levels in a way which seeks to minimize the variance of the final estimate. In the context of this algorithm, a level is the value of the score function $\Phi(\omega)$, whose purpose is to help discriminate good execution paths from bad ones with respect to a given property.

Each score function is problem-dependent and is usually crafted manually to guide the ISpl process. We see this as a limitation and make an attempt in proposing a general form of the score function for all probabilistic systems which can be modeled as HMMs. This general function calculates the scores based on the results of two independent processes, described so far:

1. The *offline* reachability analysis performed on the DTMC M. We have seen that the result of this analysis is an ordering of states based on their potential for satisfying the property φ and captured by DFA D.
2. The *online* state estimation with ISam, described in Section 4, which provides the score function with the probability distribution of the HMM M states during each step of the ISpl process.

In addition to these two fundamental sources of information, the score function may also benefit from timing information in case the property to check is time-bounded (e.g. $\mathbf{F}^{1.5s}eat$).

The general form of the score function Φ for an execution path ω is given by the following formula:

$$\Phi(\omega) = \left(\sum_{i=1}^{N_h} \sum_{j=1}^{N_d} w_{i,j}(t) \cdot L_{i,j} \right) \cdot \Delta\Phi(t) \tag{1}$$

where $L_{i,j}$ is the level of the DTMC state (i,j), and $w_{i,j}(t) = P(X_t = x_i, S_t = s_j)$ is the probability that, at time t, the system HMM is in state x_i and the property DFA is in state s_j, as estimated by the ISam. To account for the duration of different execution paths, $\Delta\Phi(t) \in [0,1]$ rewards paths with a shorter t and penalizes those with a longer t. For time-independent properties, $\Delta\Phi(t) = 1$.

Intuitively, the score is a weighted average of precalculated levels $L_{i,j}$, whereby the value of each level is weighted with the probability that, at time t, the system has reached that particular level.

It is easy to show that $\Phi(\omega) \in [0,1]$. Namely, since for any level $L_{i,j}$ we know that $L_{i,j} \in [0,1]$, and $w_{i,j} \in [0,1]$, from (1) we have

$$0 \leq \sum_{i=1}^{N_h} \sum_{j=1}^{N_d} w_{i,j}(t) \cdot L_{i,j} \leq \sum_{i=1}^{N_h} \sum_{j=1}^{N_d} w_{i,j}(t) \tag{2}$$

Notice that the term on the right-hand side of inequality (2) is the sum of the probabilities $P(X_t = x_i, S_t = s_j)$ in the parallel composition of the system HMM with the property DFA, and is thus equal to 1. Considering the fact that $\Delta\Phi(t) \in [0,1]$, from (1) and (2), it follows that $\Phi(\omega) \in [0,1]$. □

7 Experimental Results

To investigate the behavior of FC-SSC in case of Dining Philosophers and the Incrementer-or-Resetter, we performed multiple experiments on a PC computer with a 3.0 GHz dual-core Intel® Pentium® G2030 CPU and 4 GB of RAM, running Linux. In the preparatory phase, we first executed both programs for an extended period of time, collecting the traces of emitted symbols. These traces were subsequently used with UMDHMM [16] to learn an HMM for each of the two programs. These HMMs are shown in Figures 4 and 5, respectively.

7.1 Dining Philosophers

We have executed the program with 100 threads in order to find, within a short time T, the probability that a particular one of them satisfies the property $\varphi = \mathbf{F}^T eat$. We repeated the experiment for different values of T, varying from 1 to 3 seconds. The results are summarized in Figure 7.

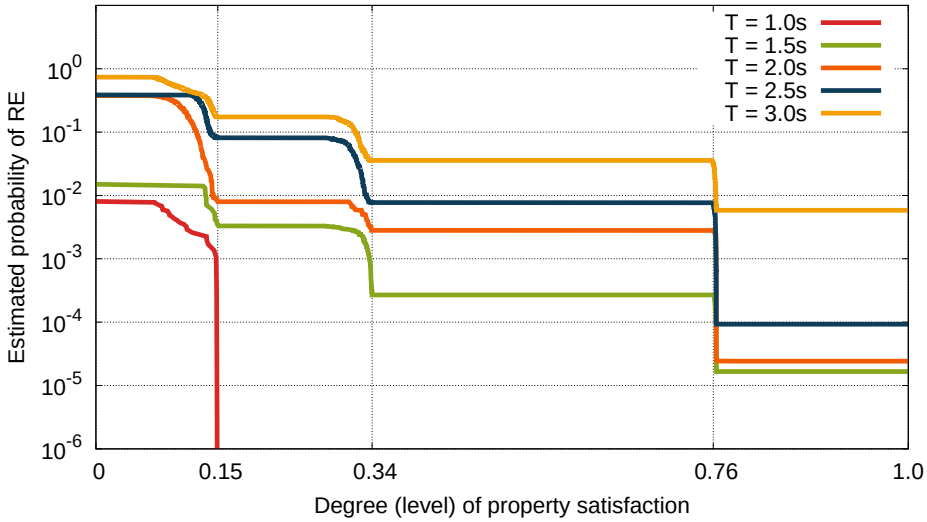

Figure 7: FC-SSC in action. Shown is the process of estimating the probability of RE expressed by the temporal property $\varphi = \mathbf{F}^T eat$ for different values of T in the Dining Philosophers program with $N = 100$ threads. ISpl was run with 1,000 traces and ISam used 280 particles for state estimation.

In Fig. 7, we can observe that in the case of $T = 1s$, even with a fairly large number of samples ISpl was not able to cross the first level boundary. This is not a failing of the ISpl process, rather, it can be attributed to the fact that the startup time of the Dining Philosophers program takes a big fraction of this 1 second. Thus, it is difficult to observe any events at all from the program in

such a short time, no matter how many samples are used. A rigorous timing analysis may find that observing the *eat* event from any philosopher within the first second is impossible.

7.2 Incrementer-or-Resetter

As in the case of Dining Philosophers, we first defined the property based on the RE whose probability we want to estimate. In this case, the property to check was $\varphi = \mathbf{F}(counter = 12)$. Given the fact that the probability of incrementing the counter at each step is 2^{-1}, we know that the theoretical probability p_φ of reaching the value of 12 is 2^{-12}, which is approximately $2.44 \cdot 10^{-4}$.

We repeated an experiment, which consisted of 3,000 ISpl traces, 12 times to estimate p_φ, and obtained the following results:

0.00000708, 0.00000570, 0.00003328, 0.00064780, 0.00000568, 0.00000282,
0.00060066, 0.00007848, 0.00000136, 0.00004132, 0.00001440, 0.00000611

For the results shown, we obtained a mean of $1.204 \cdot 10^{-4}$ with a variance of $5.599 \cdot 10^{-8}$, which provides a very good estimate of p_φ. Moreover, in most cases, we end up with a significant underestimation of p_φ which provides a lower bound on this probability. Note that a Monte Carlo experiment, for the same amount of time, would have likely produced a zero estimator with no clue about the states to visit in order to satisfy the property.

7.3 Discussion

It is interesting to note that there are several critical points in the ISpl process at which the probabilities fall significantly. Incidentally, these critical points correspond to the levels calculated by PRISM in the initial reachability analysis and shown in Fig. 6. Between these levels, the scoring function guides the ISpl process slowly forward, by discarding only the traces with the very lowest score. As such, the traces with the best potential (i.e. the highest scores) will be brought to the level boundary. If there is a critical mass of traces with scores greater than the level boundary, these will be multiplied through resampling and enable the ISpl process to continue towards its intended destination, which is the satisfaction of the property. If, on the other hand, only a small number of traces cross the level boundary, chances are that the ISpl process will be left with a degenerate set of traces all having the same score, in which case no further progress can be made.

Our results collectively show that FC-SSC typically provides a very good approximation of the actual probability and addresses the difficult CPS problem of steering a program along unlikely but successful paths with respect to an RE property. It also, as observed in the case of Incrementer-or-Resetter, can be used to provide a lower bound $\tilde{p}$ such that the system in question likely satisfies the qualitative property $P(\varphi \mid H) \geq \tilde{p}$.

8 Conclusions

In this paper, we introduced *feedback-control statistical system checking*, or FC-SSC for short, a new approach to statistical model checking that exploits principles of feedback-control for the analysis of cyber-physical systems. To the best of our knowledge, FC-SSC is the first statistical system checker to efficiently estimate the probability of rare events in realistic CPS applications or in any complex probabilistic program whose model is either not available, or is infeasible to derive through static-analysis techniques.

FC-SSC is also a new and intuitive approach for combining importance sampling (ISam) and importance splitting (ISpl) as two distinct components of a feedback controller. ISam and ISpl were originally developed for the same purpose, viz. rare event (RE) estimation. With FC-SSC, we have shown how they can be synergistically combined.

A key component of our current approach is that we learn an HMM model of a representative process (or thread) of the system we are attempting to verify. We then compose this HMM with the DFA of the property under investigation to obtain an DTMC, which we then subject to level-set analysis. The benefit of this approach is that the representative process is small enough to render the HMM-learning process and subsequent analysis readily tractable, as we have carefully avoided the pitfalls of state explosion. The price to paid in doing so is that the level-set analysis is performed on a local process-level basis, possibly resulting in an increase in the number of particles that must be considered in the subsequent importance-sampling phase.

Our current approach can be extended by considering a representative process and its *neighborhood of influence*, thereby extending the range of the level-set analysis, while still carefully avoiding state explosion. For example, in the case of Dining Philosophers, one could consider a representative process P_i and its direct neighbors P_{i-1} and P_{i+1}. One could then extend this neighborhood in a bounded fashion by considering neighbors of neighbors, while still avoiding state explosion. This line of investigation will be a focus of our future work.

References

1. Code repository. https://ti.tuwien.ac.at/tacas2015/.
2. M. Barbara, D. Frédéric, R. Gerhard, L. Alain, J. Frans, and P. Thierry, editors. *Parallel Computing: From Multicores and GPU's to Petascale, Proceedings of the conference ParCo 2009, 1-4 September 2009, Lyon, France*, volume 19 of *Advances in Parallel Computing*. IOS Press, 2010.
3. E. Bartocci, R. Grosu, A. Karmarkar, S. Smolka, S. D. Stoller, E. Zadok, and J. Seyster. Adaptive runtime verification. In *Proc. of RV 2012, the third International Conference on Runtime Verification, September, 2012 Istanbul, Turkey*, volume 7687 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2012.
4. M. Broy and E. Geisberger. Cyber-physical systems, driving force for innovation in mobility, health, energy and production. *Acatech: The National Academy Of Science and Engineering*, 2012.

5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
6. E. Clarke and P. Zuliani. Statistical model checking for cyber-physical systems. In *Proc. of ATVA 2011: the 9th International Symposium on Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 1–12. Springer, 2011.
7. A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.
8. M. Duflot, L. Fribourg, and C. Picaronny. Randomized dining philosophers without fairness assumption. *Distributed Computing*, 17(1):65–76, 2004.
9. P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel Splitting for Estimating Rare Event Probabilities. *Oper. Res.*, 47(4):585–600, 1999.
10. R. Grosu and S. Smolka. Monte Carlo model checking. In *Proc. of TACAS'05, the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *LNCS*, pages 271–286, Edinburgh, Scotland, April 2005. Springer Verlag.
11. C. Jegourel, A. Legay, and S. Sedwards. Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *LNCS*, pages 327–342. Springer, 2012.
12. C. Jegourel, A. Legay, and S. Sedwards. Importance Splitting for Statistical Model Checking Rare Properties. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 576–591. Springer-Verlag, 2013.
13. C. Jegourel, A. Legay, and S. Sedwards. An effective heuristic for adaptive importance splitting in statistical model checking. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II*, pages 143–159, 2014.
14. H. Kahn and T. E. Harris. Estimation of Particle Transmission by Random Sampling. In *Applied Mathematics*, volume 5 of *series 12*. National Bureau of Standards, 1951.
15. K. Kalajdzic, E. Bartocci, S. Smolka, S. Stoller, and G. Grosu. Runtime Verification with Particle Filtering. In *Proc. of RV 2013, the fourth International Conference on Runtime Verification, INRIA Rennes, France, 24-27 September, 2013*, volume 8174 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 2013.
16. T. Kanungo. UMDHMM tool. http://www.kanungo.com/software/software.html.
17. L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. 77(2):257–286, February 1989.
18. S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, 11(2):305–345, February 1999.
19. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 3rd edition, 2010.
20. S. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. Smolka, and E. Zadok. Runtime Verification with State Estimation. In S. Khurshid and K. Sen, editors, *RV'11 Proceedings of the Second International Conference on Runtime Verification*, volume 7186 of *LNCS*, pages 193–207. Springer, 2012.
21. V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis [robot fault diagnosis]. *Robotics Automation Magazine, IEEE*, 11(2):56–66, 2004.
22. H. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.
23. P. Zuliani, C. Baier, and E. Clarke. Rare-event verification for stochastic hybrid systems. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 217–226. ACM, 2012.